基于Clang的算法题基础 -clang

- 一、 Clang基础
  - 1. 关键字
  - 2. 特殊结构
  - 3. 先调用再实现
  - 4. sizeof()
  - 5. `void func(int val, int* val, int &val)`
  - 6. 常用变量名
- 二、 数据类型
  - 1. 字符串
  - 2. 指针
  - 3. 一维数组
  - 4. 二维数组
  - 5. 哈希表
- 三、 库函数
  - <string.h>
    - 1. 比较列表是否相等 -- memcmp(列表1, 列表2, 长度)
    - 2. 拷贝/追加 -- memcpy(列表1, 列表2, 长度)
    - 3. 初始化 -- memset(列表1, 值, 长度)
    - 4. 找到匹配字符位置 -- memchr(列表1, 待查字符, 长度)
  - <stdio.h>
  - <math.h>
    - 1. 最大/最小 -- fmax(数字1, 数字2） / fmin(数字1, 数字2）
  - <limits.h>
    - 1. `INT_MIN` and `INT_MAX` 表示int类型的上下限
- 四、 工具函数
  - 1. 快排 -- qsort(待排序数组, 数组元素个数, 每个元素大小, cmp)

一、 Clang基础

1. 关键字

1. 真假
   - 真 -> `true` 或 `1`

    - 假 -> false 或 2
2. 与或
    - 与 -> &&
    - 或 -> ||
3. 空 NULL
4. 输入输出
    - %s 字符串
    - %c 字符
    - %d 整数
    - %f 浮点数
    - %p 指针

2. 特殊结构

1. 三元表达式

```
int num = true ? 1 : 0;
```

3. 先调用再实现

```
int main() {
    // 声明
    void f(int , char); // 也可以 void f(int b, char str2);

    // 调用
    f(1, 'h');
    return 0;
}

void f(int a, char str) {
    printf("world\n");
}
```

4. sizeof()

1. 简述：
    - 指针类型： 8字节（无论指针、指针数组、指针字符、指针字符串）
    - int类型： 4字节
    - char类型：1字节
2. 示例：

```c
#include <stdio.h>

void f(int arr2[20], int arr3[], int* arr4, int * arr5) {
    // 数组作为参数传来时，被退化为指针(char类型同理)
    printf("\n-------------fstart-----------------\n");
    printf("sizeof(int) = %lu\n", sizeof(int));     // 4
    printf("sizeof(arr2) = %lu\n", sizeof(arr2));   // 8 数组作为参数传给函数时，是传给数
    printf("sizeof(arr3) = %lu\n", sizeof(arr3));   // 8 报错，同上
    printf("sizeof(arr4) = %lu\n", sizeof(arr4));   // 8 不报错
    printf("sizeof(arr5) = %lu\n", sizeof(arr5));   // 8 不报错
    printf("\n-------------fend-----------------\n");
}

int main()
{
    int arr1[10] = {1, 2, 3, 4};
    int* p_arr1 = arr1;

    printf("sizeof(int) = %lu\n", sizeof(int));         // 4
    printf("sizeof(arr1) = %lu\n", sizeof(arr1));       // 40
    printf("sizeof(int*) = %lu\n", sizeof(int*));       // 8
    printf("sizeof(p_arr1) = %lu\n", sizeof(p_arr1));   // 8
    // 参数传递数组，被退化为指针
    f(arr1, arr1, arr1, p_arr1);

    char str1[10] = "abcd";
    char* p_str1 = "abcd";

    printf("sizeof(char) = %lu\n", sizeof(char));       // 1
    printf("sizeof(str1) = %lu\n", sizeof(str1));       // 10
    printf("sizeof(char*) = %lu\n", sizeof(char*));     // 8
    printf("sizeof(p_str1) = %lu\n", sizeof(p_str1));   // 8

    return 0;
}
```

5. void func(int val, int* val, int &val)

1. 简述：
   对于整数值
   - `int val` --> 传值，不改变原值
   - `int* val` --> 传值地址，原值协同改变

   对于数组
   - `int val, int* val` --> 最终都会变为指针数组

   对于 `int &val` ，这是Cpp的引用方法，Clang不适用。
2. 示例：

```c
#include <stdio.h>

void PrintList (int* s) {
    for(int i=0; i<10; i++) {
        printf("%d  ", s[i]);
    }
    printf("\n");
}

void intUse1(int a) {
    a = 1;
}

void intUse2(int* a) {
    *a = 6;
}

// 这是CPP的“引用”，不适用于Clang
// void intUse3(int &a) {
//      a = 3;
// }

void listUse1(int arr[]) {
    arr[1] = 1;
}

void listUse2(int* arr, int index) {
    arr[index] = index;
}

int main() {
    int a = 0;
    printf("a的地址 = %p  \n", &a);
    printf("a的值 = %d  \n", a);
    printf("---------int-------------\n");

    intUse1(a);
    printf("void intUse1(int a)  --> %d  \n", a);

    intUse2(&a); // 将其地址传入
    printf("void intUse2(int* a)  --> %d  \n", a);


    printf("----------list-----------\n");
    int arr[10];
    memset(arr, 0, sizeof(arr));
    int* p_arr = arr;

    listUse1(arr);      // 也可以传指针listUse1(p_arr); 如下面调用方式二：传指针数组
    PrintList(arr);
```

```
    // 调用方式一：直接传数组
    listUse2(arr, 2);
    PrintList(arr);
    // 调用方式二：传指针数组
    listUse2(p_arr, 3);
    PrintList(arr);
    PrintList(p_arr);

    return 0;
}
```

Output:

```
a的地址 = 0x7ffeebab7458
a的值 = 0
---------int-------------
void intUse1(int a)  --> 0
void intUse2(int* a)  --> 6
----------list------------
0  1  0  0  0  0  0  0  0  0
0  1  2  0  0  0  0  0  0  0
0  1  2  3  0  0  0  0  0  0
0  1  2  3  0  0  0  0  0  0
```

6. 常用变量名

变量	解释	变量	解释
temp	临时变量	cur / curr / pos / index	当前
slow	快指针	fast	慢指针
high	上界	low	下界
start	开始	end	结束
prev / prior	前趋	post	
res / ret	返回内容	cnt	count计数器
rest	剩余	len / length	长度
row	行	col	列
top	栈顶	S / stack	某个栈
front	队首	rear	队尾

变量	解释	变量	解释
Q / queue	队列	root	树的根结点
val / ch	一个值/字符	arr	数组

二、 数据类型

1. 字符串

1. 定义和初始化
   - 方法一： 将**字符串**视为**数组**

     ```
     char str1[7] = {'a', 'b', 'c', 'd', 'e', 'f', '\0'};
     puts(str1); // Output: abcdef
     ```

   - 方法二： 法一的简化

     ```
     char str2[] = "abcdef";
     ```

2. 指针

1. 指针的定义和初始化

   ```
   int var = 10;
   int *p;     // 或 int *p = &var;
   p = &var;

   printf("地址 = %p = %p \n", p, &var);
   printf("值 = %d = %d \n", *p, var);
   ```

   Output:

   ```
   地址 = 0x7ffeec7d3464 = 0x7ffeec7d3464
   值 = 10 = 10
   ```

2. 指针的运算

   ```
   int var = 10;
   int *p = &var;
   printf("%d", ++(*p));   // 不要写*(p++)
   ```

   Output:

3. 一维数组

1. 数组
    1. 定义和初始化
        - 方法一： 定义同时初始化

```
int arr1[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
```

        - 方法二： 先定义再初始化

```
int arr2[10];
for (int i = 0; i < 10; i++) {
    arr2[i] = i+1;
}
```

        - 方法三：**数组指针**转化为**数组**

          见 `<string.h> memcpy` 函数
2. 指针数组
    1. 定义和初始化
        - 方法一：由**数组**转化为**指针数组**

```
int arr1[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int* p_arr1 = arr1;
```

        - 方法二：直接定义空**指针数组**

```
int* p_arr2 = malloc(sizeof(int) * 10);
for (int i = 0; i < 10; i++) {
    p_arr2[i] = i+1;
}
```

4. 二维数组

1. 二维数组的定义和初始化
    - 方法一： 先定义再出初始化

```
int row = 2, col = 3;
int arrs[row][col];
memset(arrs, 0, sizeof(int)*row*col);  //也可用两层for

// 打印
for (int i = 0; i < row; i++) {
    for (int j = 0; j < col; j++) {
        printf("%d  ", arrs[i][j]);
    }
    printf("\n");
}
```

- 方法二： 定义同时初始化

```
int row = 2, col = 3;
int arrs[2][3] = {1, 2, 3, 4, 5, 6};
// 等效于 int arrs[2][3] = {{1, 2, 3}, {4, 5, 6}};

// 打印
for (int i = 0; i < row; i++) {
    for (int j = 0; j < col; j++) {
        printf("%d  ", arrs[i][j]);
    }
    printf("\n");
}
```

易错注意：

- 对于方法二，使用

```
int row = 2, col = 3;
int arrs[row][col] = {1, 2, 3, 4, 5, 6};    // 错误error: variable-sized object
```

是错误的，因为数组的大小不能用变量表示的；
可以在开头定义 `#define N 100` ， 然后在代码中使用常量N

```
int arrs[N][N] = {1, 2, 3, 4};
```

2. 二维指针数组
   - 方法一：定义同时初始化

```c
int row = 2, col = 3;
// 定义和申请空间
int** pp_arrs = malloc(sizeof(int*) * row);
for (int i = 0; i < row; i++) {
    pp_arrs[i] = malloc(sizeof(int) * col);
    // 初始化--动态分配一行，memset()一行
    memset(pp_arrs[i], -1, sizeof(int) * col);
    // 如果需要初始化为其他数字，则需要用两层for赋值
}
// 打印结果
for (int i = 0; i < row; i++) {
    for (int j = 0; j < col; j++) {
        printf("%d  ", pp_arrs[i][j]);
    }
    printf("\n");
}
```

易错提示：

- 不可以定义和申请空间完后一起 `memset()` 整个二维指针数组，因为不同时间申请的空间不连续。【详见 `memset()` 部分易错提示】

5. 哈希表

> 见 `<uthash.h>` 相关

三、 库函数

参数表示说明：

- index：表示下标
- len：表示数组可含元素的个数 或 字符串中字符个数
- size_len：表示数组的内存字节数 = sizeof(elemType) * len = (若非指针）sizeof(arr)
- arr123[]：表示数组
- arr123[row][col]：表示二维数组
- *p_arr123：表示指针数组
- **p_arr123：表示二维指针数组
- str123：表示字符串
- *str123：表示指针字符串

<string.h>

1. 比较列表是否相等 -- memcmp(列表1, 列表2, 长度)

```
int res = memcmp(arr1/*arr1/str1/*str1, arr2/*arr2/str2/*str2, size_len);
```

1. 说明：
   比较 列表1 和 列表2，前size_len字节的元素是否相同；
   - 若 列表1 = 列表2， return 0;
   - 若 列表1 < 列表2， return < 0;
   - 若 列表1 > 列表2， return > 0;
2. 示例：

```
int arr1[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int arr2[10]=  {1, 2, 3, 4, 5, 6, 7, 9, 10};

char str1[4] = "abcd";
char str2[4] = "abcd";

int res = memcmp(str1, str2, sizeof(str1));
printf("res = %d\n", res);
```

2. 拷贝/追加 -- memcpy(列表1, 列表2, 长度)

```
memcpy(arr1/*arr1/str1/*str1, arr2/*arr2/str2/*str2 + len, size_len);
```

或

```
memcpy(&arr1[index]/&*arr1[index]/&str1[index]/&*str1[index], arr2/*arr2/str2/*str2 + len, size_len);
```

1. 说明：
   将 列表2 的第len下标开始，共size_len字节的内容 **拷贝/追加** 至 列表1。
   易错注释：
   - 易错点1:
     注意拷贝内容大小不能超过 列表1 的内存大小；
   - 易错点2:
     列表1 为指针字符串时，不能指向为在 **全局静态区**，否则报错 `/bin/sh: line 1: 15613 Bus error: 10` ；
     如下面代码时错误的：

```
char* p_str1 = "abcd";

puts(p_str1);
memcpy(&p_str1[4], p_str1+1, sizeof(char)*4);
puts(p_str1);
```

应该改为：

```
char str1[10] = "abcd";
char* p_str1 = str1;

puts(p_str1);
memcpy(&p_str1[4], p_str1+1, sizeof(char)*4);
puts(p_str1);
```

- 易错点3:
  若字符串仅声明，但没有初始化，直接用 `memcpy()` 会导致没有 `\0`

2. 示例：
   - 基础拷贝功能

```
int arr1[10] = {1, 2, 3, 4};
int arr2[10]=  {11, 12, 14};
// 拷贝前:
for (int i = 0; i < 10; i++) {
    printf("%d ", arr1[i]);     // Output: 1 2 3 4 0 0 0 0 0 0
}
printf("\n");

memcpy(arr1, arr2, sizeof(int)*3); // sizeof(int)*3 = 拷贝4*3字节大小的内容

// 拷贝后:
for (int i = 0; i < 10; i++) {
    printf("%d ", arr1[i]);     // Output: 11 12 14 4 0 0 0 0 0 0
}
```

   - 指定拷贝内容

```
int arr1[10] = {1, 2, 3, 4};
int arr2[10]=  {11, 12, 14};

// 拷贝前:
for (int i = 0; i < 10; i++) {
    printf("%d ", arr1[i]);     // Output: 1 2 3 4 0 0 0 0 0 0
}
printf("\n");

memcpy(arr1, arr2+1, sizeof(int));  // 从 arr[1]开始拷贝，只拷贝sizeof(int) = 4字节

// 拷贝后:
for (int i = 0; i < 10; i++) {
    printf("%d ", arr1[i]);     // Output: 12 2 3 4 0 0 0 0 0 0
}
```

   - 追加 / 指定拷贝内容存放位置

```
int arr1[10] = {1, 2, 3, 4};
int arr2[10]=  {11, 12, 14};

// 拷贝前:
for (int i = 0; i < 10; i++) {
    printf("%d ", arr1[i]);     // Output: 1 2 3 4 0 0 0 0 0 0
}
printf("\n");

memcpy(&arr1[4], arr2, sizeof(int)*3);  // 在arr1[4]开始追加

// 拷贝后:
for (int i = 0; i < 10; i++) {
    printf("%d ", arr1[i]);     // Output: 1 2 3 4 11 12 14 0 0 0
}
```

- 指针数组示例

```
int arr1[10] = {1, 2, 3, 4};
int arr2[10]=  {11, 12, 14};
int* p_arr1 = arr1;
int* p_arr2 = arr2;

for (int i = 0; i < 10; i++) {
    printf("%d ", p_arr1[i]);       // Output: 1 2 3 4 0 0 0 0 0 0
}
printf("\n");
memcpy(&p_arr1[4], p_arr2+1, sizeof(int));  // 不要用sizeof(int*)

for (int i = 0; i < 10; i++) {
    printf("%d ", p_arr1[i]);     // Output: 1 2 3 4 12 0 0 0 0 0
}
```

- 字符串示例

```
char str1[10] = "abcd";
char str2[10] = "mnpq";

puts(str1);     // Output: abcd
memcpy(&str1[4], str2+1, sizeof(char)*4);
puts(str1);     // Output: abcdnpq
```

- 指针字符串示例

```
char str1[10] = "abcd";
char str2[10] = "mnpq";

char* p_str1 = str1;
char* p_str2 = str2;

puts(p_str1);       // Output: abcd
memcpy(&p_str1[4], p_str2+1, sizeof(char)*4);

puts(p_str1);       // Output: abcdnpq
```

3. 初始化 -- memset(列表1, 值, 长度)

```
memcmp(arr1/*arr1/str1/*str1, value, size_len);
```

1. 说明:
   将 列表1 中的前size_len字节初始化为value值
   注意:
   - 若value为数字， 则只能为 -1 或 0;
   - 若value为字符， 则单字符即可;

   易错提示:
   - `memset()` 是对连续的 `size_len` 字节进行初始化，但是对于二维数组或者二维指针数组可能列与列之间 `malloc()` 时不连续， 所以会导致越界的问题。
2. 示例:
   - 数组:

     ```
     int arr1[10] = {1, 2, 3, 4};
     for (int i = 0; i < 10; i++) {
         printf("%d ", arr1[i]);     // Output: 1 2 3 4 0 0 0 0 0 0
     }
     printf("\n");
     memset(&arr1[3], -1, sizeof(int)*5);
     for (int i = 0; i < 10; i++) {
         printf("%d ", arr1[i]);     // Output: 1 2 3 -1 -1 -1 -1 -1 0 0
     }
     ```

   - 指针数组:

```
int arr1[10] = {1, 2, 3, 4};
int* p_arr1 = arr1;
for (int i = 0; i < 10; i++) {
    printf("%d ", p_arr1[i]);     // Output: 1 2 3 4 0 0 0 0 0 0
}
printf("\n");
memset(&arr1[3], -1, sizeof(int)*5);
for (int i = 0; i < 10; i++) {
    printf("%d ", p_arr1[i]);     // Output: 1 2 3 -1 -1 -1 -1 -1 0 0
}
```

- 字符串

```
char str1[10] = "abcd";

puts(str1);     // Output: abcd
memset(&str1[2], '$', 4);
puts(str1);     // Output: ab$$$$
```

- 字符串数组

```
char str1[10] = "abcd";
char* p_str1 = str1;    // 注意不要指向全局静态域

puts(p_str1);     // Output: abcd
memset(&p_str1[2], '$', 4);
puts(p_str1);     // Output: ab$$$$
```

4. 找到匹配字符位置 -- memchr(列表1, 待查字符, 长度)

```
char* res = memchr(str1/*str1, ch, size_len);
```

1. 说明：
   找到 字符ch 在 列表1 的前size_len字节第一次出现位置，并返回对应位置及其后续的字符串；
2. 示例：

```
char ch = 'c';
char *ret;

char str[] = "abcdefghijklmnopqrstuvwxyz";
ret = memchr(str, ch, strlen(str));
puts(ret);     // Output: cdefghijklmnopqrstuvwxyz

puts("----------------");
char* p_str = "abcdefghijklmnopqrstuvwxyz";
ret = memchr(p_str, ch, sizeof(char)*26);
puts(ret);     // Output: cdefghijklmnopqrstuvwxyz
```

<stdio.h>

<math.h>

1. 最大/最小 -- fmax(数字1, 数字2）/ fmin(数字1, 数字2）

<limits.h>

1. `INT_MIN` and `INT_MAX` 表示int类型的上下限

四、 工具函数

1. 快排 -- qsort(待排序数组, 数组元素个数, 每个元素大小, cmp)

> 实现对 数组 / 指针数组 / 字符串 / 指针字符串 的快速排序

1. 参数
   - 待排序数组： int arr[] / int* arr / char str[10] / char* str
   - 数组元素个数：list_len
   - 每个元素大小：sizeof(int)
   - 函数名称：cmp
2. 示例：

```c
int cmp(int* a, int* b) {
    return *a - *b; // 若需要降序改为 *b - *a
}

int main() {
    int arr1[10] = {1, 4, 3, 6, 5, 0, 7, 6, 9, 10};
    int len = sizeof(arr1) /  sizeof(arr1[0]);

    qsort(arr1, len, sizeof(int), cmp);

    // Output:
    for (int i = 0; i < 10; i++) {
        printf("%d ", arr1[i]);
    }

    return 0;
}
```